Software Quality for AI: Where we are now?

Valentina Lenarduzzi[1], Francesco Lomio[2], Sergio Moreschini[2], Davide Taibi[2], and Damian Andrew Tamburri[3]

[1] LUT University, Lahti, Finland
valentina.lenarduzzi@lut.fi,
[2] Tampere University, Tampere, Finland
[francesco.lomio;sergio.moreschini;davide.taibi]@tuni.fi,
[3] Eindhoven University of Technology - JADS, 's-Hertogenbosch, The Netherlands
d.a.tamburri@tue.nl

Abstract. Artificial Intelligence is getting more and more popular, being adopted in a large number of applications and technology we use on a daily basis. However, a large number of Artificial Intelligence applications are produced by developers without proper training on software quality practices or processes, and in general, lack in-depth knowledge regarding software engineering processes. The main reason is due to the fact that the machine-learning engineer profession has been born very recently, and currently there is a very limited number of training or guidelines on issues (such as code quality or testing) for machine learning and applications using machine learning code. In this work, we aim at highlighting the main software quality issues of Artificial Intelligence systems, with a central focus on machine learning code, based on the experience of our four research groups. Moreover, we aim at defining a shared research road map, that we would like to discuss and to follow in collaboration with the workshop participants. As a result, the software quality of AI-enabled systems is often poorly tested and of very low quality.

Keywords: Software Quality, AI Software

1 Introduction

The term Artificial Intelligence (AI) commonly indicates a software system that is capable of mimicking human intelligence [27]. AI systems are capable of performing actions thanks to underlying algorithms that can learn from the data without being specifically programmed. The set of these algorithms are referred to as Machine Learning (ML) algorithms [21].

As any software system, AI systems require attention attaining quality assurance, and in particular to their code quality [26]. Conversely, current development processes, and in particular agile models, enable companies to decide the technologies to adopt in their system in a later stage and it becomes hard to anticipate if a system, or if a data pipeline is used for Machine-Learning (ML) produces high-quality models [23].

The need for considering the quality of AI-enabled systems was highlighted already even more than 30 years ago [31] [26]. For the time being, different approaches have been proposed to evaluate the quality of the AI-models but little in the way of AI code quality itself (e.g. [15] and [8]).

Conversely, as already mentioned, the overall quality of the AI-enabled systems, and in particular the ML code has never been investigated systematically so far if not anecdotally. For example, a report from Informatics Europe[4] and the ACM Europe Council[5], as well as the Networked European Software and Services Initiative[6], highlighted the importance of assessing the quality of AI-related code [16] [23]. The EU has also proposed a whitepaper discussing a high-level approach to the regulatory compliance of AI, but did not focus on code quality issues at all [12] .

AI code needs to be maintained. Therefore, developers need to take care of the quality of their code, and keep the technical debt [6] under control [28].

The goal of this paper is to highlight the quality-related issues of AI software, as well as possible solutions that can be adopted to solve them. The identification of such quality issues is based on the experience of our four research groups: (1) the Software Engineering group of the LUT University, (2) the Machine Learning and (3) Software Engineering groups of the Tampere University, and the (4) Jheronimus Academy Data and Engineering (JADE) lab of the Jheronimus Academy of Data Science.

The insights in this paper enable researchers to understand possible problems on the quality of AI-enabled systems opening new research topics and allows companies to understand how to better address quality issues in their systems.

The remainder of this paper is structured as follows. Section 2 presents Related works. Section 3 described the current issues on code quality of AI-enabled systems, Section 4 proposes our shared road map while Section 5 draws conclusions.

2 Related Work

As any software system, AI-enabled software, and in general ML code, require to pay attention to quality assurance, and in particular to the code quality. Current development processes, and in particular agile models, enable companies to decide the technologies to adopt in their system in a later stage. Therefore, it is hard to anticipate if a system, or if a data pipeline used for ML will produce high-quality models [23].

A limited number of peer-reviewed works highlighted the quality issue in AI-enabled software.

Murphy et al. [22] proposed a testing framework for Machine Learning (ML), introducing the concept of regression testing and an approach for ranking the cor-

[4] Informatics Europe `https://www.informatics-europe.org`

[5] ACM Europe Council `https://europe.acm.org`

[6] The Networked European Software and Services Initiative - NESSI `http://www.nessi-europe.com`

rectness of new versions of ML algorithms. Besides the proposed model, they also highlighted conflicting technical terms with very different meanings to machine learning experts than they do to software engineers (e.g. “testing”, “regression”, “validation”, “model”). Moreover, they raised the problem of code quality, reporting that future works should address it. Related to the matter of ML testing, Zhang et al. provided a comprehensive survey research [30]. In this work with the term ML testing, they refer to *”any activity aimed at detecting differences between existing and required behaviors of machine learning systems.”* The work comprehends a section related to fundamental terminology in ML which will be referred to in Table 1.

Nakajima, in his invited talk, call the attention the product quality of ML-based systems, identifying new challenges for quality assurance methods. He proposed to identify new testing methods for ML-based systems, proposing to adopt Metamorphic testing [10] is a pseudo oracle approach and uses golden outputs as testing values.

Pimentel et al. [10] investigated the reproducibility of Jupyter notebooks, showing that less than 50% of notebooks are reproducible, opening new questions to our community to propose advanced approaches for analyzing Jupyter notebooks. Wang et al [29] analyzed 2.5 Million of Jupiter notebooks investigating their code quality reporting that notebooks are inundated with poor quality code, e.g., not respecting recommended coding practices, or containing unused variables and deprecated functions. They also report that poor coding practices, as well as the lack of quality control, might be propagated into the next generation of developers. Hence, they argue that there is a strong need to identify quality practices, but especially a quality culture in the ML community.

The vast majority of grey literature also focuses on the quality of the ML models or on the data. Only a limited number of authors raised the problem of the overall product quality or of the quality of the ML code. Vignesh [24] proposed to continuously validate the quality of ML models considering black boxes techniques and evaluating the performance of model post-deployment on test data sets and new data from production scenarios. He also proposes to adopt metamorphic testing, involving data mutation to augment test data sets used for evaluating model performance and accuracy.

It is interesting to note that Vignesh recommends exposing models being tested as RESTful service, instead of testing internally or manually. As for the quality of the code of ML Models, Dhaval [20] proposes to introduce code review processes for ML developers, adopting code reviewing techniques traditionally adopted in SW Engineering.

Besides the model itself, the essence of a good machine learning-based software relies on the data used to train the network. It is therefore vital to take into account the characteristics and features that mark a specific application and meet such qualities in the data used to develop it. Hence, some of the requirements that the data needs to meet are: context compatibility, incorruptibility, and completeness. In a data-driven software scenario, it is not rare to find to encounter a situation in which the same data set is used to train networks with different goals. As an example, most of the networks generated in computer vision are fine-tuned over a first tuning on ImageNET [13]. In such a situation it is very important to take into account the *compatibility* between the context for which the data has been created and the one we are trying to use the data for. With *incorruptibility* we define the quality of a data set to be resistant to external factors that might generate errors, or undesired changes, during writing, reading, storage, transmission, or processing.

A data set is *complete*, related to a specific application when it is capable of describing and defining specific information (in the form of mono or multidimensional variables) in its entirety. Related to a Machine Learning-based approach we say that the data set is complete when it is capable of tuning the weights to generate the desired result without any requirement for fine-tuning. As an example, we take the MNIST data set [17] which is complete if we are training a network to understand handwritten digits, but not in the case when we want to train a network to understand handwriting as it does not include letters. To this matter, an ulterior data set has been created, known as EMNIST [11].

Lwakatare et al performed the work closest to this work [19]. They discussed software engineering challenges based on a survey conducted on four companies. The result is a taxonomy of challenges that consider the life-cycle activities of ML code (such as assemble data set, create model, (re)train and evaluate, deploy). However, differently than in our work, they did not identify clear issues and possible solutions.

3 AI Software Quality: Key Issues and Comments

Based on the collective experience of our groups and through simple self-ethnography [9], we elicited different code quality issues commonly faced by all sorts of stakeholders (e.g., our research assistant working for consultancy projects in AI, our colleagues, and our students as developers of AI Software) working with AI software. In this Section, we describe the aforementioned elicitation, also discussing possible solutions that might be adopted to solve the emerged issues.

– **Developers Skills and Training**. Once the suitable data has been chosen and proven to be compliant with our requirements the next step involves coding. The machine learning engineer profession was born less than a decade ago and therefore, no training guidelines have been outlined yet. Most of the professionals that occupy this position have been moving from a similar field such as mathematics, physics, or computer vision. This grouping of different backgrounds generated "communication problems" which reflected in the way the code was written. We identify four open problems nested in this macro area: code understandability, code quality guidelines, training problems, and the absence of tools for software quality improvement. One of the main issues of AI Developers is the lack of skills in software engineering, and especially the lack of skills in software quality and maintenance. The reason is that AI developers are often experts borrowed from other fields such as physics or mathematics. The lack of skills in software quality is then reflected in their code. Moreover, as highlighted by Wang et al. [29], the AI code examples often provided online by AI experts and teachers are of low quality, mainly because they lack in software engineering skills. As a result, these poor coding practices may further be propagated into the next generation of developers. For these reasons, there has been a rise in the number of courses created to instruct specific approaches related to AI in different fields [14].

- **Development Processes**. Because of the aforementioned lack of skills and training in software engineering, AI developers often lack knowledge of development processes. As a result, in our experience, it is often hard to introduce them into running agile processes, while it is much easier to introduce them into a waterfall process. The main reason is their problem in segmenting the work into smaller units or user stories, and to incrementally deliver the results, as usually done in agile processes.
- **Testing Processes**. Testing AI code is usually considered for AI developers as testing and training the machine learning algorithms. The introduction of unit and integration testing is of paramount importance to ensure the correct execution of software systems. However, the different testing practices, usually applied in software engineering, are not commonly applied to the AI-related code, but only to the overall systems including them, and the AI-specific function is not commonly part of the CI/CD pipeline. The reason might be in the complexity of running tests, and the problem of the non-deterministic results of the AI-algorithms. Metamorphic testing can be a good solution, at least to run unit testing.
- **Deployment confidence**. Developers commonly prefer to manually test their AI-models, and then to manually deploy the systems in production, mainly because they are not confident in releasing every change automatically. The reason is still connected to the lack of clear testing processes and the last of integration of unit and integration tests.
- **Code Quality**. AI-code is often of very low quality, mainly because of the lack of quality guidelines, quality checks, and standards. We might expect the IDEs to highlight possible "code smells" in the code, or to highlight possible styling issues. However, the current support from IDEs is very limited.
  Specific libraries, such as Tensorflow, refer to the document PEP-8 [25], which is the official style guide for Python code. However, the latter does not take into account the specific pipeline which involves the different stages of developing AI-code. An interesting initiative is conducted by Pytorch, which relies on Pytorch [4], a wrapper developed to organize the code in sections and separates the research code (dataloader, optimizer, training step) from the engineering code (i.e. training process). Even if Pytorch does not provide clear styling guidelines, it pushes developers to adopt clear guidelines for structuring the code.
- **Incompatibility of new version of ML libraries**. A well known problem in ML, is the necessity of installing a specific version of libraries as most of those might not be compatible with their future releases mining the success

of the final project. This often creates inconsistency and incompatibility of the old version of the system with newer versions [1] [2] [5].A possible workaround is to use microservices, adopting a specific version of a library on a service, and eventually another version of the same library on another service. However, when the existing code needs to be executed on a newer version of the same library, migration issues need to be considered.
– **Code Portability**. The rise of multiple libraries for ML training together with the different background of the engineers generated a new *"Babel tower"* for coding. Libraries such as Pytorch or Tensorflow have a different backbone which makes the same application look very different from one another. Therefore, to understand and port the code from one library to the other, it is often nontrivial.
– **Terminology**. AI and Software Engineering usually adopts the same terms for different purposes. This issue usually creates misunderstandings between developers with different backgrounds. In order to clarify the terms adopted by both domain, in Table 1 we present and describe the most common misleading terms together with their meaning in AI and in SW Engineering. Some terms have a totally different meaning in the two domains, while others might be used for the same purpose in different contexts. As an example, the term "parameter", besides the different definitions proposed in Table 1, is in both cases used to describe inputs or properties of objects for configuration. As another example, the term "code" can be also used in AI to describe the set of instructions to build the different layers, to recall the input dataset, and to perform training, test (and when necessary validation).
– **Communication between AI Developers and other developers** . Because of the different terminology adopted, we commonly experienced communication challenges between AI and other developers. As an example, we often had communication issues with AI developers without a software engineering background, especially when discussing scalability, architectural, or development processes related issues.

4 Research Roadmap

In order to address the issues reported in Section 3, we would like to share a collaborative research road map, that we are aiming at investigating in collaboration with different groups.

– Identify the most common quality issues in AI code using traditional SW Engineering methodologies. In this step, we are aiming at conducting a set of empirical studies among AI-developers and end-users of AI-based systems, intending to elicit the most important internal quality issues experienced by the developers and the external issues perceived by the users of the same systems.
– Identify a set of testing techniques applicable to AI-enabled systems, to allow their execution into the CI/CD pipeline, and increase their confidence in the deployment.

Table 1. The terminology adopted in AI and in SW Engineering

Term	**Machine Learning**	**Software Engineering**
Class	"One of a set of enumerated target value for a label" resulted from a classification model [3]	An extensible template definition for instantiating objects in object-oriented programming.
Code	The possible values of a field (variable), also known as "category"	The source code
Distribution	Probability distribution (from statistics). Sometimes also referred to distributed computing or parallelism.	Distributed computing. In testing, refers to the testing of distributed systems.
Example	An entry from a dataset, composed of at least one features and a label, that can be used in model training and/or after training during inference. (also known as Observation)	An example system or piece of software often found in software documentation.
Execution Environment	The set of all the libraries which are installed, and lately imported in the compiler. Specifically, in reinforcement learning, is the observable world exposed to a learning agent [3]	A system comprised of hardware and software tools used by the developers to build/deploy software.
Feature	A characterizing variable found in the input data to a machine learning model. Predictions about the data can be made after gaining insight from these features (training).	A distinguishing characteristic of a software item
Function	A mathematical function, mapping parameters to a domain	In principle the same, in practice refers to the implementation in the source code
Label	In supervised learning: expected output for a training case. For example, an email message may be labeled as SPAM or non-SPAM. [3]	A label refers to the name of a text field in a form or user interface (e.g the label of a button).
Layer	A set of neurons in a neural network that operates on input data (possibly from previous layers) and provides inputs to the next layer(s) as their outputs. [3]	A layer in a multilayered software solution, e.g. data access layer, business logic layer, presentation layer.
Model	Output of a ML algorithm after it has been trained from the training data, the learning program, and frameworks [30]	Different meaning, depending on the context: Development Model (process), Data Model (Database schema), ...
Network	Usually assuming Neural Network	Usually assuming Computer Network
Parameter	A variable in a model that is adjusted during training to minimize loss. E.g. weights or normalization parameters.	A variable that holds a piece of data provided to a function as an input argument.
Pattern	Detecting patterns in a dataset.	Design patterns or architectural patterns
Performance	How well a certain model performs according to the selected metrics, e.g. precision, recall, or false positives, after training [21].	How fast a certain piece of software executes and/or how efficient it is.
Quality	See performance	Software quality, including internal (e.g. code quality) and external quality (e.g. usability, performance, ...)
Reference	The baseline category used to compare with other categories.	The variable that points to a memory address of another variable
Regression	Estimate numerical values, identifying relationships, and correlations between different types of data [30].	Regression (Testing) is a full or partial selection of already executed test cases which are re-executed to ensure that a recent program or code change has not adversely affected existing features.
Testing	The process of evaluating a network over a set of data which has not been used for training and/or validation	Check whether the actual results match the expected results and to ensure that the software system is Defect free
Training	The process of tuning the weights and biased of a network recursively by making use of labeled examples	Developer's training
Validation	The process of using data outside the training set, known as the validation set, to evaluate the model quality. Important to ensure the generalization of the model outside the training set [3]	Verification and validation is the process of checking that a software system meets specifications and that it fulfills its intended purpose (requirements).

- Identify a set of quality rules for the code and develop a linter for detecting code-level issues for the most common libraries and languages, starting from widely used tools for static code analysis [18] and technical debt analysis [7].
- Integrate into our master courses of Machine Learning a course on software engineering, with a special focus on the maintenance and testing of the AI-enabled applications

5 Conclusion

In this work, we highlighted the most common quality issues that our developers face during the development of AI-enabled systems, based on the experience of our three research groups.

Overall, the training of developers is one of the biggest lacks in AI, which usually brings several issues related to low code quality of AI-code as well as low long-term maintenance.

Acknowledgement

We would like to thank the software engineering and AI community and in particular Taher Ahmed Ghaleb, Steffen Herbold, Idan Huji, Marcos Kalinowski, Christian Kästner, Janet Siegmund and Daniel Strüber for helping us on the definition of the glossary of AI and SW Engineering terms[7].

References

1. TensorFlow version compatibility. https://www.tensorflow.org/guide/versions
2. Compatible Versions of PyTorch/Libtorch with Cuda 10.0. `https://discuss.pytorch.org/t/compatible-versions-of-pytorch-libtorch-with-cuda-10-0/58506` (2019), online; accessed 11 July 2020
3. Machine Learning Glossary, Google Developers. https://developers.google.com/machine-learning/glossary (2019), online; accessed 28 August 2020
4. Pytorch Lightning. The lightweight PyTorch wrapper for ML researchers. `https://github.com/PyTorchLightning/pytorch-lightning` (2019), online; accessed 11 July 2020
5. Tensorflow 1.11.0 incompatible with keras2.2.2? https://github.com/tensorflow/tensorflow/issues/22601 (2019), online; accessed 11 July 2020
6. Avgeriou, P., Kruchten, P., Ozkaya, I., Seaman, C.: Managing technical debt in software engineering (dagstuhl seminar 16162). Dagstuhl Reports 6 (01 2016)
7. Avgeriou, P., Taibi, D., Ampatzoglou, A., Arcelli Fontana, F., Besker, T., Chatzigeorgiou, A., Lenarduzzi, V., Martini, A., Moschou, N., Pigazzini, I., Saarimäki, N., Sas, D., Soares de Toledo, S., Tsintzira, A.: An overview and comparison of technical debt measurement tools. IEEE Software (2021)

[7] `https://twitter.com/vale_lenarduzzi/status/1295055334264975360` Last access: 28 August 2020

8. Bradley, A.P.: The use of the area under the roc curve in the evaluation of machine learning algorithms. Pattern recognition 30(7), 1145–1159 (1997)
9. Britten, N., Campbell, R., Pope, C., Donovan, J., Morgan, M., Pill, R.: Using meta ethnography to synthesise qualitative research: a worked example. J Health Serv Res Policy 7(4), 209–215 (Oct 2002), http://www.ncbi.nlm.nih.gov/pubmed/12425780
10. Chen, T.Y.: Metamorphic testing: A simple method for alleviating the test oracle problem. In: Proceedings of the 10th International Workshop on Automation of Software Test. p. 53–54. AST '15, IEEE Press (2015)
11. Cohen, G., Afshar, S., Tapson, J., Van Schaik, A.: Emnist: Extending mnist to handwritten letters. In: 2017 International Joint Conference on Neural Networks (IJCNN). pp. 2921–2926. IEEE (2017)
12. Commission, E.: WHITE PAPER On Artificial Intelligence - A European approach to excellence and trust. https://ec.europa.eu/info/sites/info/files/commission-white-paper-artificial-intelligence-feb2020_en.pdf?utm_source=CleverReach&utm_medium=email&utm_campaign=23-02-2020+Instituts-Journal+07\%2F20\%3A+Wo+waren+Sie\%3F+Es+ging+um+Sie\%21&utm_content=Mailing_11823061 (2020), online; accessed 09 July 2020
13. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: 2009 IEEE conference on computer vision and pattern recognition. pp. 248–255. Ieee (2009)
14. Kästner, C., Kang, E.: Teaching software engineering for ai-enabled systems. arXiv preprint arXiv:2001.06691 (2020)
15. Kohavi, R.: A study of cross-validation and bootstrap for accuracy estimation and model selection. In: Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2. p. 1137–1143. IJCAI'95, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1995)
16. Larus, J., Hankin, C., Carson, S.G., Christen, M., Crafa, S., Grau, O., Kirchner, C., Knowles, B., McGettrick, A., Tamburri, D.A., Werthner, H.: When computers decide: European recommendations on machine-learned automated decision making (2018)
17. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE 86(11), 2278–2324 (1998)
18. Lenarduzzi, V., Sillitti, A., Taibi, D.: A survey on code analysis tools for software maintenance prediction. In: Software Engineering for Defence Applications — SEDA (2019)
19. Lwakatare, L.E., Raj, A., Bosch, J., Olsson, H.H., Crnkovic, I.: A taxonomy of software engineering challenges for machine learning systems: An empirical investigation. In: Agile Processes in Software Engineering and Extreme Programming. pp. 227–243. Springer International Publishing (2019)
20. M., D.: How to perform Quality Assurance for Machine Learning models? https://medium.com/datadriveninvestor/how-to-perform-quality-assurance-for-ml-models-cef77bbbcfb (2018), online; accessed 09 July 2020
21. Mitchell, T.M.: Machine learning. McGraw-Hill, New York, NY [u.a. (2010), http://www.amazon.com/Machine-Learning-Tom-M-Mitchell/dp/0070428077
22. Murphy, C., Kaiser, G.E., Arias, M.: A framework for quality assurance of machine learning applications. Columbia University Computer Science Technical Reports, CUCS-034-06 (2006)
23. Nessi: Software and Artificial Intelligence. http://www.nessi-europe.com/files/NESSI\%20-\%20Software\%20and\%20AI\%20-\%20issue\%201.pdf (2019), online; accessed 09 July 2020

24. Radhakrishnan, V.: How to perform Quality Assurance for Machine Learning models? `https://blog.sasken.com/quality-assurance-for-machine-learning-models-part-1-why-quality-assurance-is-critical-for-machine-learning-models` (2019), online; accessed 09 July 2020
25. van Rossum, G., Warsaw, B., Coghlan, N.: PEP 8 – Style Guide for Python Code. https://www.python.org/dev/peps/pep-0008/
26. Rushby, J.: Quality measures and assurance for ai (artificial intelligence) software. Tech. rep. (1988)
27. Russell, S.J., Norvig, P.: Artificial intelligence - a modern approach: the intelligent agent book. Prentice Hall series in artificial intelligence, Prentice Hall (1995)
28. Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.F., Dennison, D.: Hidden technical debt in machine learning systems. In: Advances in neural information processing systems. pp. 2503–2511 (2015)
29. Wang, J., Li, L., Zeller, A.: Better code, better sharing:on the need of analyzing jupyter notebooks (2019)
30. Zhang, J.M., Harman, M., Ma, L., Liu, Y.: Machine learning testing: Survey, landscapes and horizons. IEEE Transactions on Software Engineering (2020)
31. Ören, T.I.: Quality assurance paradigms for artificial intelligence in modelling and simulation 48(4), 149–151 (1987)